Extending libbpf for Kubernetes

Vinay Kulkarni

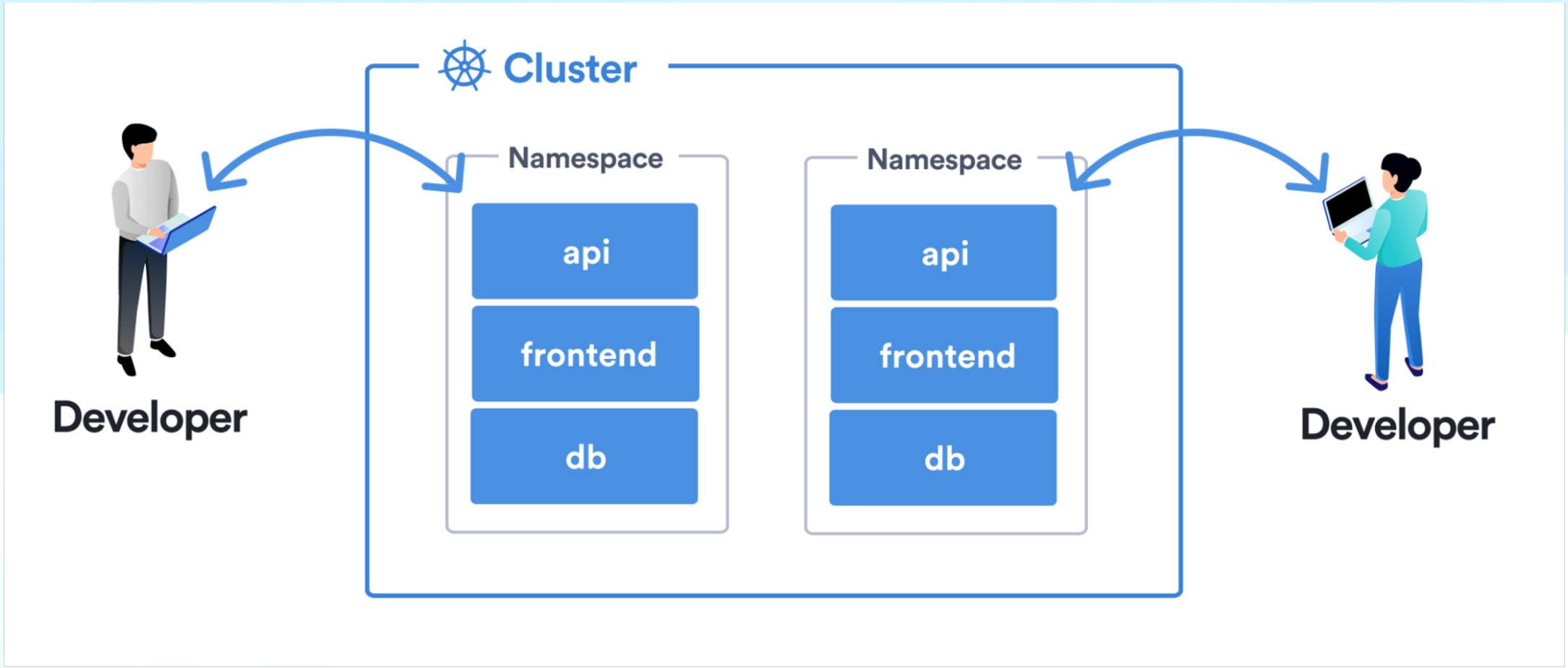
@vinaykul 

@iSkibum 

Agenda

- Cloud-based Development Environment Use Case - Okteto Inc.
  - Problem at Hand …
  - … How eBPF Came to the Rescue
  - Pain Points
  - Proposed libbpf Extension
- Other Use Cases
- Discussion

Cloud-based Development Environment



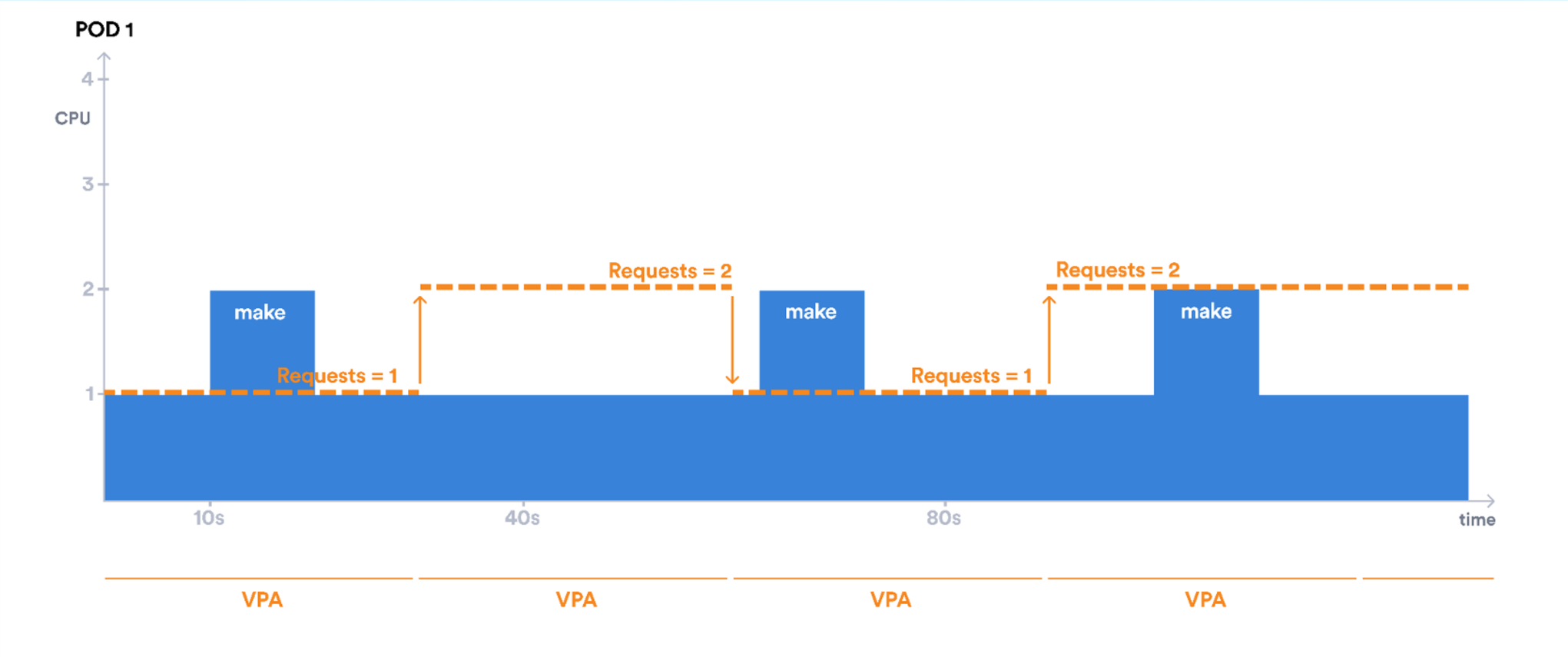
- Write code locally -> rsync -> build or run tests in K8s pod
- https://kccncna2022.sched.com/event/182HU

Cloud-based Dev Env Pod Example

- Build/Test Environment Pod Spec
  - Container: kube-build-ctr
  - Resources:
    - Reserves CPU and memory needed to build code, run a battery of tests
- Until recently, resources reserved in K8s pod was static
    - Once scheduled and running, it cannot be changed without restart

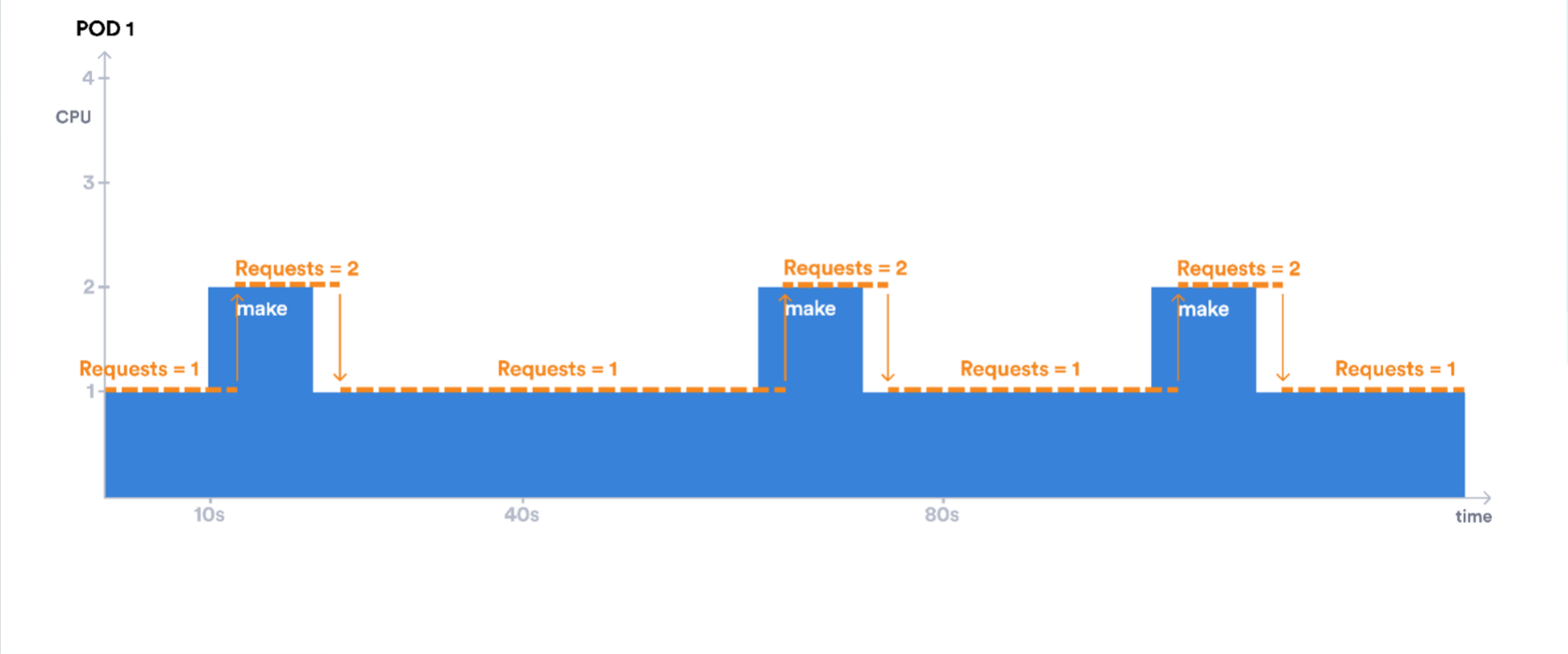
```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: kube-build-pod
5   spec:
6     containers:
7     - name: kube-build-ctr
8       image: skiibum/kube-build-arm64:v1.25
9       imagePullPolicy: IfNotPresent
10      command: ["tail", "-f", "/dev/null"]
11      resources:
12        limits:
13          cpu: "5"
14          memory: "5Gi"
15        requests:
16          cpu: "4"
17          memory: "5Gi"
18
```

Problem is …



- Latest Kubernetes (v1.27) enables in-place restart-free resize of pod resources (CPU, memory)
- Vertical Pod Autoscaler is a tool that can resize pod resources based on usage
  - Reactive - may not be good enough! (OOM kills)

Ideally …



- Pod resources are resized before it becomes a problem
  - Proactive

eBPF makes it possible!

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kube-build-pod
spec:
  containers:
  - name: kube-build-ctr
    image: skiibum/kube-build-arm64:v1.25
    imagePullPolicy: IfNotPresent
    command: ["tail", "-f", "/dev/null"]
    resources:
      limits:
        cpu: "5"
        memory: "50Mi"
      requests:
        cpu: "4"
        memory: "50Mi"
```

```python
# podsnoop.py: Prototype eBPF program that snoops on pod exec activity
#              requires linux-headers, bpfcc-tools, kubectl
#              To run: sudo python3 podsnoop.py
import os
from bcc import BPF

POD_SNOOP_eBPF_CODE = r"""TRACEPOINT_PROBE(syscalls, sys_enter_execve) {
    char task_cmd[32];
    bpf_get_current_comm(&task_cmd, sizeof(task_cmd));
    bpf_trace_printk("Launching program: %s\n", task_cmd);
    return 0;
}"""
bpf = BPF(text = POD_SNOOP_eBPF_CODE)

while True:
    try:
        (task, pid, cpu, flags, ts, msg) = bpf.trace_fields()
        if str.__contains__(msg.decode("utf-8"), "make"):
            pod_name = os.popen("nsenter -t %s -u hostname 2>/dev/null" % pid).read().strip()
            if pod_name == "kube-build-pod":
                patch_str = '{"spec":{"containers":[{                                    \
                                "name":"kube-build-ctr",                                 \
                                "resources":{"requests":{"memory":"5Gi"},"limits":{"memory":"5Gi"}} \
                              }]}}'
                patch_cmd = "kubectl patch pod %s --patch '%s' > /dev/null" % (pod_name, patch_str)
                os.system(patch_cmd)
    except ValueError:
        continue
```

- action = (command == ‘make’) ? resize pod : have a beer ;)

Some Rough Edges

- Trace (via perf_event) commands (e.g make) only for the container (cgroup_id) we care about

```
root@vbuild:~#
root@vbuild:~# kubectl create -f ~/YML/kube-build-pod.yaml
pod/kube-build-pod created
root@vbuild:~#
root@vbuild:~# kubectl get pod kube-build-pod -ojson | jq .status.containerStatuses[0].containerID
"containerd://fd4078a980e9fc4ce9124b8a96f8da377c9a15a9ce87345e7660016e9cb4e7c1"
root@vbuild:~#
root@vbuild:~# find /sys/fs/cgroup/ -name fd4078a980e9fc4ce9124b8a96f8da377c9a15a9ce87345e7660016e9cb4e7c1
/sys/fs/cgroup/kubepods/pod2a9c7101-d341-4f68-8587-9b510274bece/fd4078a980e9fc4ce9124b8a96f8da377c9a15a9ce87345e7660016e9cb4e7c1
root@vbuild:~#
root@vbuild:~# ls -ladi /sys/fs/cgroup/kubepods/pod2a9c7101-d341-4f68-8587-9b510274bece/fd4078a980e9fc4ce9124b8a96f8da377c9a15a9ce8
7345e7660016e9cb4e7c1 | awk '{print $1}'
7008
root@vbuild:~#
```

```
localhost:~ # kubectl create -f ~/YML/kube-build-pod.yaml
pod/kube-build-pod created
localhost:~ # kubectl get pod kube-build-pod -ojson | jq .status.containerStatuses[0].containerID
"containerd://d049ed022df453cdaac16850d82ccf6bd9930cdc0e3f9d3622a9fe83cda605f8"
localhost:~ # find /sys/fs/cgroup/ -name d049ed022df453cdaac16850d82ccf6bd9930cdc0e3f9d3622a9fe83cda605f8 | grep cpu
/sys/fs/cgroup/cpuset/kubepods/podd54bcb54-258e-49be-be19-54e2455190ee/d049ed022df453cdaac16850d82ccf6bd9930cdc0e3f9d36
22a9fe83cda605f8
/sys/fs/cgroup/cpu,cpuacct/kubepods/podd54bcb54-258e-49be-be19-54e2455190ee/d049ed022df453cdaac16850d82ccf6bd9930cdc0e3
f9d3622a9fe83cda605f8
localhost:~ # ls -ladi /sys/fs/cgroup/cpu,cpuacct/kubepods/podd54bcb54-258e-49be-be19-54e2455190ee/d049ed022df453cdaac1
6850d82ccf6bd9930cdc0e3f9d3622a9fe83cda605f8 | awk '{print $1}'
1732
localhost:~ #
```

- Not a very trivial way to find containerID <> cgroup_id mapping

Some Rough Edges

```c
SEC("tracepoint/syscalls/sys_enter_execve")
int podsnoop(void *ctx) {
    u64 cgroup_id = bpf_get_current_cgroup_id();
    struct pod_command *val = bpf_map_lookup_elem(&resize_containers_map, &cgroup_id);
    if (val != NULL) {
        u8 is_equal = 1;
        struct pod_exec_event pxevent = {};
        bpf_get_current_comm(&pxevent.cgroup_cmd, sizeof(pxevent.cgroup_cmd));
        //TODO: Find a more efficient way. Maybe 'val->cmd' should be u64 hash
        for (u8 i = 0; i < BUF_SIZE; i++) {
            if (pxevent.cgroup_cmd[i] != val->cmd[i]) {
                is_equal = 0;
                break;
            }
            if (pxevent.cgroup_cmd[i] == '\0' || val->cmd[i] == '\0') {
                break;
            }
        }
        if (is_equal) {
            pxevent.cgroup_id = cgroup_id;
            u64 id = bpf_get_current_pid_tgid();
            pxevent.cgroup_pid = id & 0xFFFFFFFF;
            long rv = bpf_perf_event_output(ctx, &pod_exec_events, BPF_F_CURRENT_CPU, &pxe
            if (rv != 0) {
                bpf_printk("DBG: podsnoop call to bpf_perf_event_output failed. ErrCode: %
            }
        }
    }
    return 0;
}
```

- ~~Maybe add bpf_strncmp(…)?~~ NVM: It has already been added in libbpf

Proposed libbpf Helper Extensions

- Add: `u64 bpf_get_container_cgroup_id(const char *container_id)`
  - e.g: container_id = “fd4078a980e9fc4ce9124b8a96f8da377c9a15a9ce87345e7660016e9cb4e7c1
  - How: Scan /sys/fs/cgroup for container_id (For cgroups v1, look under /sys/fs/cgroup/cpu)
    - If found, return its i-node number
    - If not, return 0
- Add: `int bpf_get_cgroup_container_id(u64 cgroup_id, const char *container_id)`
  - How: ~~ `find /sys/fs/cgroup -inum <cgroup_id>

Other Use Cases

- Containerized Java application with high startup CPU requirements
    - Running time CPU usage is 1/10th the startup time CPU needs
        - Allocating too little CPU -> long startup time
        - Allocating startup requirements -> underutilized cluster
        - Need: Resize down pod quickly after startup
- eBPF network stats program attachment to pod veth
    - Attach Tx stats counter eBPF program to veth ingress in host ns
        - Trace successful completion of CNI ADD to trigger attach

Discussion / Q & A

- At least two use cases that could leverage simplified cgroup_id <> container_id helpers.
    - Is this enough justification to add the proposed helpers?
- If yes, is this the right way to do it?
    - If not, any alternative suggestions?